kurz & bündig	
Inhalt	Überprüfung einer Web-Anwendung auf Sicherheitslücken
Zusammenfassung	Die Erstellung von Web-Applikationen erfordert besondere Sicherheitsvorkehrungen. Der Compuware SecurityChecker ermöglicht eine Tool-gestützte Sicherheitsanalyse.
Quellcode mit angeliefert	nein

Veröffentlichung: 09/2006.

Compuware DevPartner SecurityChecker

In Zeiten zunehmender Internet-Angriffe sehen sich Firmen verstärkt vor dem Problem, die vertrauenswürdigen Daten ihrer Kunden zu schützen. Sicherheitsreviews kosten erheblich Zeit und Geld und werden leider häufig vor dem Hintergrund wachsenden Termindrucks aus dem Projektplan “rausoptimiert”. Ein Tool-basierter Ansatz kann helfen, die wichtigsten Probleme doch noch vor dem Deployment zu beseitigen.

Manu Carus

Der SecurityChecker der Fa. Compuware ([1]) ist eine Software für die automatisierte Sicherheitsanalyse einer ASP.NET-Applikation. Das Werkzeug identifiziert die Sicherheitslücken (“Vulnerabilities”) einer Web-Anwendung und untersucht zu diesem Zweck den Source-Code, die Laufzeitumgebung sowie die Integrität der Applikation. Die ermittelten Sicherheitslücken werden nach Härtegrad kategorisiert und in einem abschließenden Report zusammengefaßt. Dabei unterscheidet sich der SecurityChecker von anderen Vulnerability Scannern durch eine enge Integration in die Entwicklungsumgebung und in den Entwicklungsprozess. In den folgenden Abschnitten wird der Leistungsumfang des SecurityCheckers an einem Fallbeispiel demonstriert.

Sicherheitsanalyse

Die Sicherheitsanalyse einer Web-Anwendung hat die Aufgabe, bekannte Schwachstellen zu identifizieren, zu dokumentieren und Vorschläge für die Behebung von Sicherheitsproblemen zu unterbreiten. Dabei ist es nicht ausreichend, nur einen Aspekt der Anwendung zu betrachten, bspw. den Source-Code statisch zu analysieren. Stattdessen muss ein Scanning der Applikation auf unterschiedlichen Ebenen erfolgen, damit insbesondere der dynamische Laufzeitkontext erfasst werden kann.

Da manuelle Sicherheitsreviews in Bezug auf Zeit und Kosten eines Projekts sehr aufwändig sind, bietet sich ein Tool-basierter Ansatz an. Der SecurityChecker ist ein solches Werkzeug, das sich nach der Installation in Form des neuen Menupunkts “SecurityChecker” in Visual Studio integriert. Die Sicherheitsanalyse läuft dann innerhalb der Entwicklungsumgebung wie folgt ab:

Die ASP.NET-Applikation wird kompiliert. Durch Aktivierung des Menus “SecurityChecker | New Session” wird ein Optionsdialog angezeigt, in dem der Fokus der Sicherheitsanalyse festgelegt wird (Abbildung 1). Nach Klick auf die Schaltfläche “Start Analysis” wird die Startseite der Web-Anwendung in einem Browserfenster angezeigt. Nun durchläuft der Entwickler die zu analysierenden Seiten der Applikation, in dem er die Formulare der Anwendung ausfüllt und den zu analysierenden Seiten folgt. Mit dem Schließen des Browserfensters startet der SecurityChecker die Sicherheitsanalyse unter Berücksichtigung der für diese Session festgelegten Optionen. Je nach Komplexität der Anwendung kann diese Analyse einen längeren Zeitraum beanspruchen. Abschließend werden die identifizierten Sicherheitslücken zusammengefasst, grafisch aufbereitet

und interaktiv zwecks Navigation zwischen Resultat und dem zugehörigen Source-Code miteinander verlinkt. Für die Erstellung eines Berichts kann das Ergebnis der Analyse in einem Report aufbereitet werden.

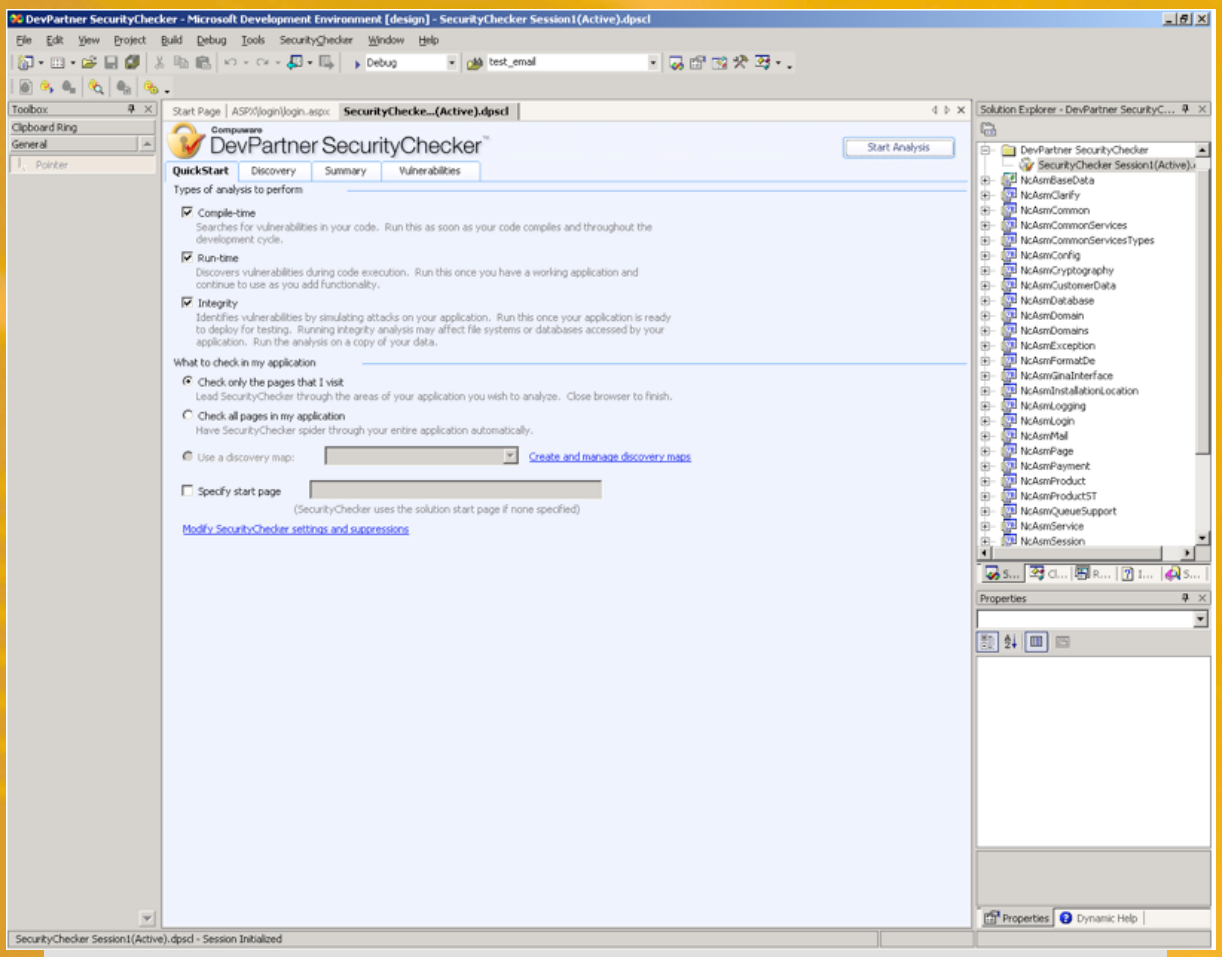

Abb. 1: Optionen des SecurityCheckers

Anwendungsbeispiel: Bestellprozess einer Portal-Applikation

Die Arbeitsweise und Leistungsfähigkeit des Tools wird im folgenden an einer Portal-Applikation demonstriert. Es handelt sich dabei um eine Web-Anwendung, die den Bestellprozess von Telefonie- und Internet-Diensten abbildet und etliche Formulare für die Eingabe der gewünschten Produkte, Konfigurationen und Kundendaten umfasst (Abbildung 2).

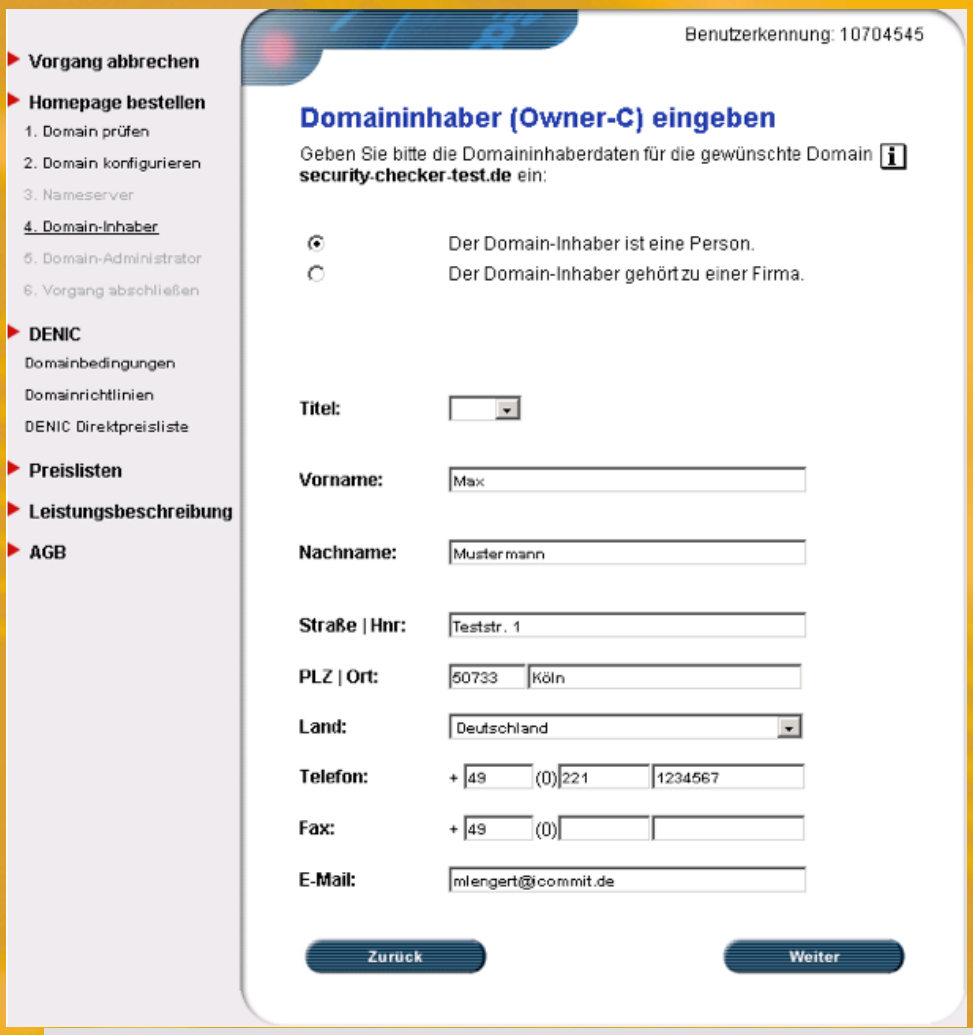

Abb. 2: Anwendungsbeispiel: ASP.NET-Applikation

Kasten 1 faßt die Kennzahlen der zu analysierenden ASP.NET-Applikation zusammen.

Kasten 1: Kennzahlen der analysierten ASP.NET-Applikation

Für die in dem Fallbeispiel untersuchte ASP.NET-Applikation wurden die folgenden Kennzahlen ermittelt:

Anzahl	Art	Datei	Lines of Code
176	Web-Controls	.ascx	84.633 LOC
241	Formulare	.aspx	40.901 LOC
158	VB.NET-Klassen in 40 Assemblies	.vb	345.321 LOC
67	C#-Klassen in 8 Assemblies	.cs	34.425 LOC
117	HTML-Seiten	.html	51.348 LOC
34	JavaScript-Dateien	.js	37.151 LOC
14	Stylesheets	.css	29.810 LOC

Vulnerability Scanning

Das Scanning der beschriebenen Portalapplikation dauert (auf einem herkömmlichen Entwicklungsrechner) 1 Stunde und 45 Minuten. Das Ergebnis zeigt Abbildung 3.

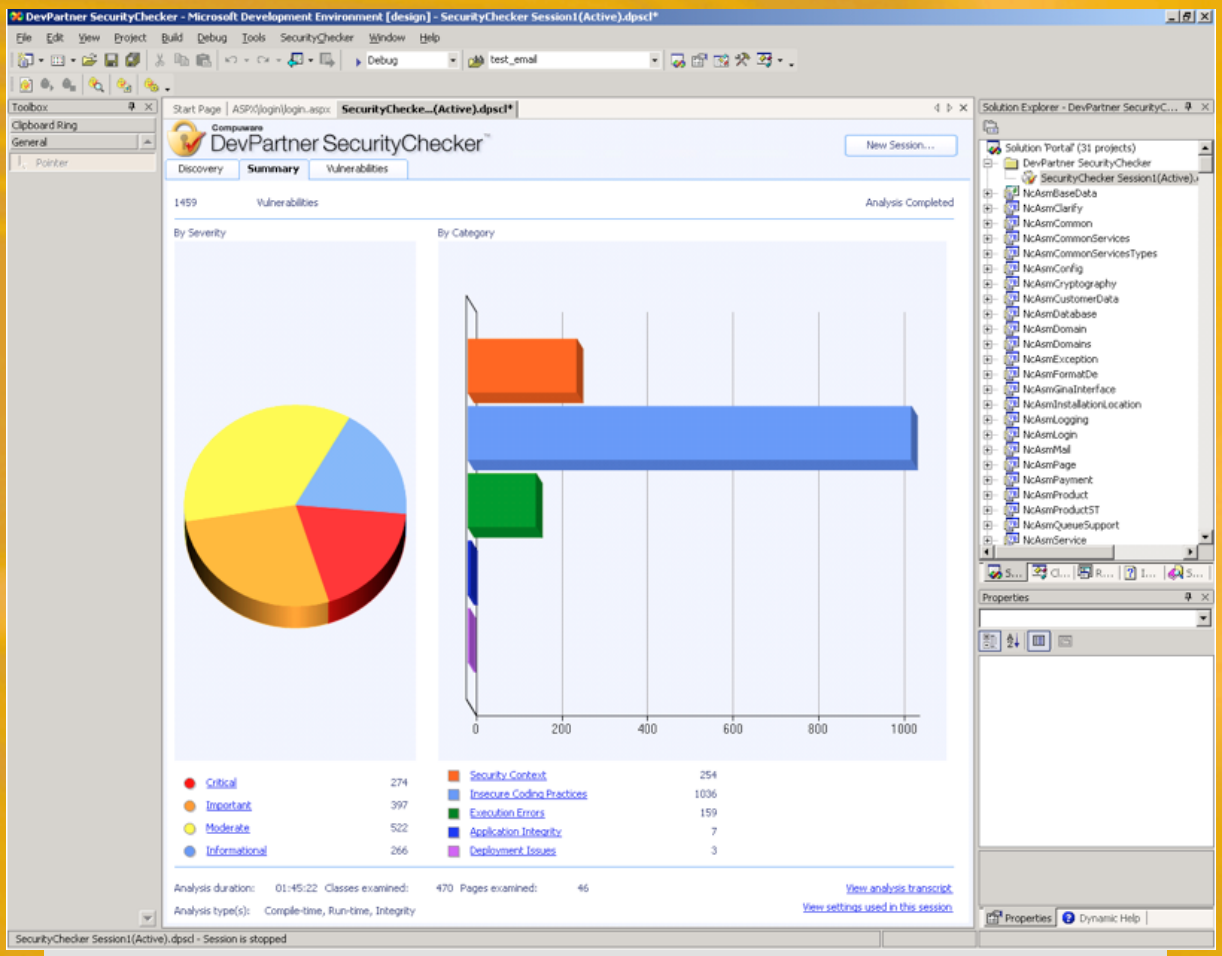

Abb. 3: Ergebnis des Vulnerability Scanning

Die identifizierten Schwachstellen werden nach Härtegrad und Kategorie unterschieden. Der Härtegrad beschreibt, ob die Sicherheitslücke ein hohes Sicherheitsrisiko darstellt oder ob die Behebung der Schwachstelle lediglich zu empfehlen ist:

- Eine Schwachstelle ist “Critical”, wenn die Applikation einem bekannten Hacking-Exploit ausgesetzt ist. Ein Exploit beschreibt ein Verfahren zur Ausnutzung einer Sicherheitslücke (bspw. einen Algorithmus oder einen URL). Kritische Sicherheitslücken beschreiben ein hohes Risiko, falls sie vor dem Deployment der Applikation nicht behoben werden.
- Als “Important” gelten Schwachstellen, die ein Angreifer zur Kompromittierung der Applikation ausnutzen könnte, so dass die Applikation einer potentiellen Attacke ausgesetzt ist.
- Mittelschwere Schwachstellen (“Moderate”) lassen für sich gesehen keine Kompromittierung einer Applikation zu. Jedoch kann ein Angreifer solche Schwachstellen in Kombination mit anderen Exploits anwenden, um abhängig von den Umgebungsbedingungen Kontrolle über die Applikation zu erlangen.
- Darüber hinaus werden Schwachstellen aufgeführt, die die Sicherheit einer Applikation durchaus beeinträchtigen können (“Informational”). Jedoch sollte der Applikationsentwickler bewerten, ob die Applikation vor ihrem Deployment von dem aufgezeigten Problem betroffen sein wird.

Die Kategorie ordnet der Schwachstelle einen Sicherheitskontext zu:

- Der “Security Kontext” betrachtet die Authentifizierungsmechanismen einer Applikation, also den Prozess, durch den die Identität des Computers oder des Netzwerks überprüft wird.
- Die Kategory “Insecure Coding Practices” umfasst bekannte Implementierungsmuster, durch die Sicherheitslücken sowohl in “managed code” als auch in “unmanaged code” entstehen.
- “Execution Errors” sind Schwachstellen und Programmierfehler, die sich erst zur Laufzeit auswirken und die Applikation einem erfolgreichen Angriff aussetzen können.
- Integrität einer Applikation (“Application Integrity”) bedeutet, wie sorgfältig die Applikation sich dem Schutz sensibler Daten widmet (bspw. Kreditkartendaten oder persönliche Daten). Eine Applikation ist integer, wenn sie wertvolle Daten schützt, indem sie eine grundsätzlich defensive Haltung bei der Verwaltung und Verarbeitung solcher Daten annimmt.
- Die Kategorie “Deployment Issues” betrachtet verschiedene Konfigurationsdetails, insbesondere die in der web.config vorgenommenen Einstellungen.

Beispiele für Sicherheitslücken der einzelnen Kategorien sind nochmals in Kasten 2 zusammengefasst.

Kasten 2: Kategorien von Schwachstellen

Die folgende Tabelle beschreibt einige Beispiele für Schwachstellen und die ihnen zugeordneten Kategorien (ohne Anspruch auf Vollständigkeit):

Kategorie	Beispiel
Security Context	• Überprüfung der Authentifizierungsmechanismen (NTLM, Forms Authentication) • fehlende oder falsch vorgenommene .NET Code Access Security • Verifikation und Fälschung von Session-Cookies • Überprüfung der minimal erforderlichen Rechte der Applikation • Scanning des aktuellen Sicherheitskontextes der Applikation • ...
Insecure Coding Practices	• öffentlich zugängliche Assemblies, die nicht "partially trusted" sind • Code, der betrügerisches Auftreten nicht erkennt oder ermöglicht • Verwendung von "unsafe" in C# • ...
Execution Errors	• unbehandelte Exceptions • fehlgeschlagene Berechtigungsanforderungen • Buffer Overflows • Cross-Site-Scripting • ...
Application Integrity Issues	• Nicht oder falsch angewandte Kryptographie kann die Integrität einer Applikation verletzen, so dass der Zugriff auf vertrauenswürdige Daten möglich wird. • Durch die Art und Weise, wie Fehler behandelt und angezeigt werden, kann sich ein Angreifer wertvolles Wissen über die Struktur der Anwendung verschaffen. • ...
Deployment Issues	• Konfiguration des Web-Servers • Konfiguration der .NET-Runtime • Sicherheit des Dateisystems (Zugriffsberechtigungen auf Dateien und Verzeichnisse) • Möglichkeiten eines Remote-Zugriffs auf die Applikation • ...

In der Tab "Vulnerabilities" werden die identifizierten Sicherheitslücken nach Härtegrad oder Kategorie gruppiert und aufgelistet (Abbildung 4). Für jeden Eintrag werden die betroffenen Dateien aufgeführt, und eine ausführliche Beschreibung erläutert, welcher Angriff angewendet wurde und welche Maßnahmen zu ergreifen sind, um sich vor derartigen Attacken zu schützen. Im Beispiel wurde ein "Parameter Tampering" durchgeführt, d.h. einzelne Eingaben der Applikation wurden mit Sonderzeichen, Hexadezimal-Code-Folgen und Buffer-Overflows "befeuert", in der Hoffnung, dass diese Eingaben von der Applikation ungeschützt verarbeitet und an die nachfolgende Softwareschicht durchgereicht werden. Ein Prozentzeichen in einem Eingabefeld kann bspw.

bewirken, dass in einem Report mehr Ergebnisse angezeigt werden, als der Benutzer eigentlich sehen dürfte, und die Eingabe eines <SCRIPT> Tags ermöglicht Cross-Site-Scripting-Attacken.

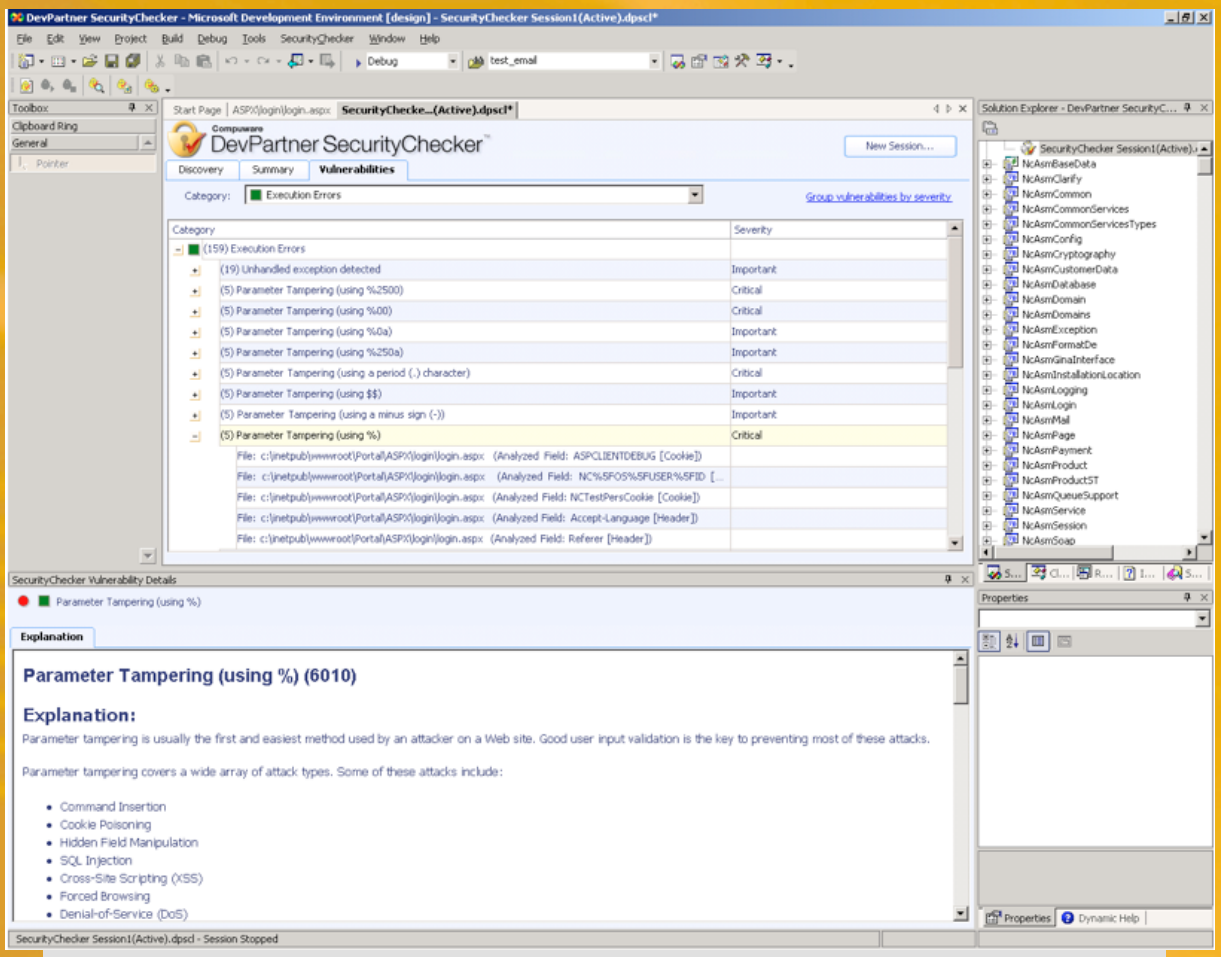

Abb. 4: Vulnerabilities

Um eine genauere Vorstellung von der Arbeitsweise des SecurityCheckers zu erhalten, werden einige beispielhafte Angriffe aufgeführt, die das Werkzeug auf die HTTP Header, Cookies, Form Fields und Query Strings der Applikation anwendet:

- URL-Redirects (z.B. "\..\..\..\..")
- Parameter-Tampering (z.B. "%","*,"&","%x%x%x%x%x", "%00/WINNT/system.ini")
- Buffer Overflows (z.B. "-9922816251426433759354 39...")
- Script Injection (z.B."%3c%73%43%52%49%50%74%3e..", was dem Anfang von "<sCRIPt>alert('...');</sCRIPt>" entspricht)
- statische Analyse aller Assemblies und Source-Codes in der ausgeführten Visual Studio-Solution

Ein ausführliches Protokoll stellt das Werkzeug in dem sog. "Analysis Transcript" zusammen, das über den Link "View analysis transcript" eingesehen werden kann (unten rechts in Abbildung 3). Listing 1 führt einige wichtige Zeilen der Sicherheitsanalyse auf.

Listing 1: Analysis Transcript

```
Opening browser to begin manual exploration of your application.
Discovery has completed successfully.
Analysis Started
Compile-time Analysis Enabled.  Launching Compile-time Analysis
Run-time Analysis Enabled.  Launching Run-time Analysis
Integrity Analysis Enabled.  Launching Integrity Analysis
Starting Code Compile-time Analysis
Starting Integrity Analysis
Run-time Analysis is resetting IIS
Run-time Analysis has completed resetting IIS
Starting Run-time Analysis
Starting replay of web requests
Processing run-time replay request number: 1
Connected to ASP.NET Worker process (ASPNET_WP.EXE) PID: 1676
Processing run-time replay request number: 2
...
Disconnected from ASP.NET Worker process (ASPNET_WP.EXE) PID: 1676
Completed Run-time Analysis
Run-time Analysis is resetting IIS
Run-time Analysis has completed resetting IIS
Beginning setup of Integrity Analyzer
Beginning setup of ASP.NET host environment
Completed setup of ASP.NET host environment in AppDomain SECCHK_4c4d4ec3
Now Analyzing Page: c:\inetpub\wwwroot\Portal\ASPX\login\login.aspx
Evaluating vulnerability vectors for page ASPX\login\login.aspx  (GET)
Adding rule check 6000 (ASPCLIENTDEBUG::\..\..\..\..\..\..\..\...) for page
ASPX\login\login.aspx
Adding rule check 6500 (ASPCLIENTDEBUG::-9922816251426433759354 39...) for page
ASPX\login\login.aspx
Adding rule check 5559 (ASPCLIENTDEBUG::<!--#exec cmd="/../../../...) for page
ASPX\login\login.aspx
Adding rule check 5000 (ASPCLIENTDEBUG::<sCRIPt>alert("A7F195CD6D...) for page
ASPX\login\login.aspx
...
Redirecting to page
http://127.0.0.1/portal/ASPX/common/checkJavascript.aspx?nctoken=88Ajqmti51qYk9opbk2WFjT50R2Jt
inV8nqNHk9UPV03d&returnurl=%2fPortal%2fASPX%2flogin%2flogin.aspx
Evaluating vulnerability vectors for page ASPX\common\checkJavascript.aspx  (GET)
Performing integrity rule checks
Running rule 6000 (ASPCLIENTDEBUG::\..\..\..\..\..\..\..\...) on page ASPX\login\login.aspx
Evaluating results of rule test 5023 for page ASPX\login\login.aspx
Finished Analyzing Page: c:\inetpub\wwwroot\Portal\ASPX\login\login.aspx
Analyzing Type Portal.NcAsmCommonServices.NcCDServices (8 of 10)
Analyzing Type Portal.NcAsmCommonServices.NcCache (9 of 10)
Analyzing Type Portal.NcAsmCommonServices.NcGlobal (10 of 10)
...
Beginning unload of host environment in AppDomain SECCHK_4c4d4ec3
Completed unload of host environment in AppDomain SECCHK_4c4d4ec3
Shutdown of Integrity Analysis has completed
Completed Integrity Analysis
Analyzing Type Portal.Anonymous.address_Label (276 of 276)
...
Starting Web Page Compile-time Analysis
Starting Web.Config Compile-time Analysis
Completed Compile-time Analysis
Analysis Completed
Analysis Statistics: Classes Examined: 470; Pages Examined: 46; Analysis Duration: 01:45:22
```

Sicherheitskontext

Sicherheit ist auf allen Ebenen einer Applikation relevant: in der eingesetzten Hardware, in der zugrunde liegenden Netzwerkstruktur, in jeder eingesetzten Software-Komponente. In jeder Schicht der Applikation: auf der Datenbank-Ebene (Tabellendesign, Stored Procedures, Benutzerberechtigungen), in den Geschäftslogik-Komponenten und auf der Präsentationsebene.

In jedem Arbeitspaket eines Projekts, beginnend mit der Spezifikation und endend mit dem Deployment und der Abnahme.

Ein Automatisierungswerkzeug kann natürlich nur die eingesetzten Software-Komponenten analysieren. Der SecurityChecker unterscheidet dabei die folgenden Analysemodi (Abbildung 1):

- Bei der "Compile-Time-Analysis" wird der Source-Code der Visual Studio-Solution einem statischen Review unterzogen. Code-Behind-Files (.cs, .vb), Web-Dateien (.aspx, .ascx, .asax, .html, .dhtml, ...) und Konfigurationsdateien (web.config) werden auf bekannte Schwachstellen und Programmiermuster untersucht, mit dem Ziel, Methoden oder Zeilen zu identifizieren, die Sicherheitslücken entstehen lassen. Da mit einer Laufzeitanalyse nur ein gewisser Abdeckungsgrad des insgesamt implementierten Codes erreicht werden kann, eignet sich diese Review-Methode insbesondere dafür, Schwachstellen zu finden, die durch die Simulation von Angriffen zur Laufzeit nur schwer oder aufwändig nachzustellen sind.
- Bei der"Run-Time-Analysis" wird die ASP.NET-Applikation einer Laufzeitanalyse unterzogen. Dabei wird bspw. überprüft, ob die Applikation exzessive Privilegien genießt, oder auf Dateien in privilegierten Verzeichnissen zugreift, auf nicht-vorgesehene Abschnitte in der Registry zugreift oder sonstige Operationen ausführt, die die Sicherheit der Applikation kompromittieren könnten.
- Die "Integrity Analysis" ähnelt am ehesten dem Leistungsumfang klassischer Vulnerability Scanner: durch das Abspielen einer Serie bekannter Sicherheitsangriffe ("Replay") wird jedes Eingabefeld der ASP.NET-Applikation "abgetastet". Headers, Cookies, Query Strings und URLs werden bekannten Angriffen wie bspw. Cross Site Scripting (XSS), SQL Injection, Buffer Overflows und Parameter Tampering ausgesetzt. Je nach Komplexität der Anwendung kann diese Analyse relativ lange dauern.

Abbildung 5 veranschaulicht, wie die einzelnen Modi bei Einsatz des SecurityCheckers in den Entwicklungsprozess einer ASP.NET-Applikation einzuordnen sind.

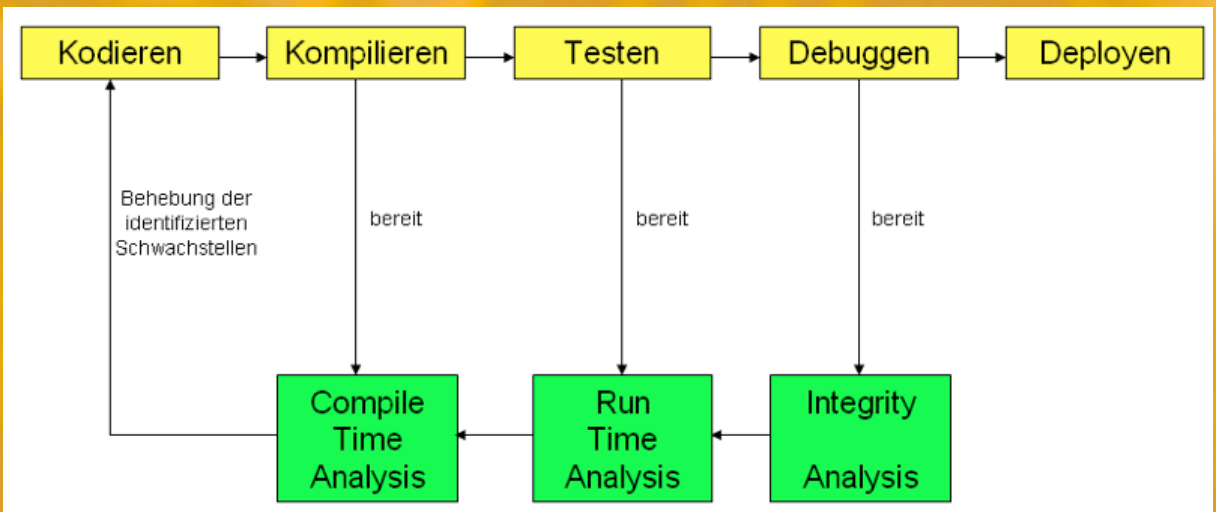

Abb. 5: SecurityChecker-Lifecycle einer ASP.NET-Applikation

Präzisierung des Tool-Einsatzes

Die oben vorgestellte Sicherheitsanalyse erfolgte unter Angabe der Option “Check only the pages that I visit” (siehe Abbildung 1). Zwar ermöglicht der SecurityChecker eine automatische Analyse aller Seiten einer ASP.NET-Applikation, indem er ausgehend von der Startseite des Projekts wie ein Robot bzw. Spider rekursiv allen Links und Buttons folgt. Jedoch wird sich jeder Entwickler bei der Sicherheitsanalyse auf seinen eigenen Verantwortungsbereich konzentrieren wollen. Um genauer einschränken zu können, welche Seiten und Eingabefelder der Applikation bei der Sicherheitsanalyse betrachtet werden sollen, kann der Entwickler eine sog. “Discovery Map” erzeugen. Darunter wird ein Abbild der zu untersuchenden Applikationsmerkmale verstanden, eine Zusammenfassung der zu besuchenden Seiten, den zu folgenden Links, den zu berücksichtigenden Controls. Enthält die Seite ein Formular, kann in der Discovery Map für jedes Eingabefeld ein Wert hinterlegt werden. Abbildung 6 zeigt eine automatisch erzeugte Discovery Map für das in Abbildung 2 dargestellte Eingabeformular.

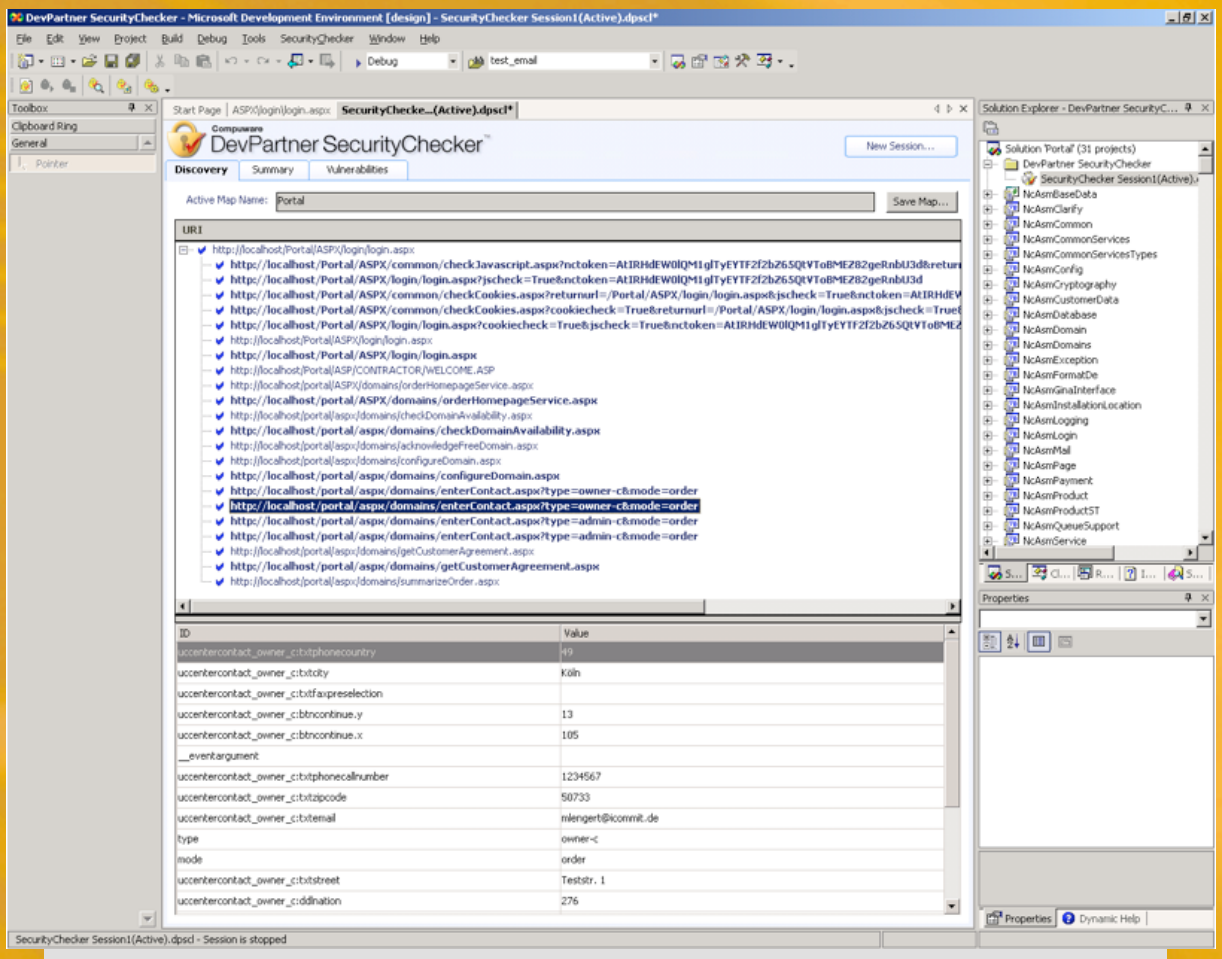

Abb. 6: Discovery Map

Darüber hinaus können sog. “Black Lists” erzeugt werden. Diese Listen enthalten Seiten, die von der automatischen Analyse ausgeschlossen werden sollen. Mit Hilfe von “Suppressions” können bestimmte Regeln und ganze Regel-Sets von der Überprüfung ausgeschlossen werden, bspw. um falsche Ergebnisse aus den Reports auszuschließen (“False Positives”) oder bestimmte Sicherheitsaspekte vorerst außer Acht zu lassen.

Das Ergebnis einer Sicherheitsanalyse kann in unterschiedlichen Reports zusammengefasst und aufbereitet werden. Der “Technical Detail Report” umfasst alle Einzelheiten und Informationen, die zu den identifizierten Schwachstellen bekannt sind, insbesondere Hilfestellungen und Erläuterungen für das Fixing. Der Report kann durch Angabe zahlreicher Optionen angepasst werden, bspw. um nur bestimmte Analysemodi anzuwenden (Compile-Time, Run-Time, Integrity), um nur bestimmte Härtegrade zu berücksichtigen (Critical, Important, Moderate, Informational) oder das Ergebnis auf bestimmte Kategorien einzuschränken (Security Context, Insecure Coding Practices, Execution Errors, Application Integrity, Deployment Issues). Der “Summary Report” umfasst weniger detaillierte Informationen und fokussiert sich stattdessen auf die visuelle Darstellung von High-Level-Informationen. Die Reports werden in einer HTML-Datei aufbereitet. Darüber hinaus können die Analyseergebnisse in einem rohen XML-Format exportiert und mit Hilfe eines eigenen XSLT-Stylesheets transformiert und individuell aufbereitet werden.

Sicherheitsanalysen können mit Hilfe eines Kommandozeilen-Interfaces automatisiert werden. Auf diese Weise ist eine Integration des SecurityCheckers in einen “Daily Build” oder in automatisierte Qualitätssicherungsprozesse möglich. Basis dieser Automatisierung ist eine XML-Datei, in der die einzelnen Optionen für den Analyselauf spezifiziert werden. Durch die Verwendung unterschiedlicher Optionsdateien können mehrere Reports mit jeweils unterschiedlichem Fokus erstellt werden.

Kritik

Der SecurityChecker ist auf den Entwicklungsprozess von ASP.NET-Applikationen zugeschnitten. Die Integration in Visual Studio ist nahtlos, die Handhabung der identifizierten Schwachstellen und deren Behebung ist einfach, gut dokumentiert und für Entwickler ausreichend Technik-lastig. Positiv fällt insbesondere auf, dass der Entwickler bei einem Doppelklick auf eine Vulnerability zu der Zeile bzw. Methode in dem betroffenen Source-Code geführt wird (siehe Abbildung 4). Umgekehrt werden dort allerdings die identifizierten Schwachstellen nicht farblich gekennzeichnet, wie es bei anderen Tools des Herstellers üblich ist (z.B. FaultSimulator).

Die Leistungsfähigkeit eines Vulnerability Scanners muss daran gemessen werden, ob alle bekannten Vulnerabilities ausreichend überprüft werden, und ob der Software-Hersteller die erforderlichen Updates fortlaufend und schnellstmöglich bereitstellt.

Laut Report überprüft der SecurityChecker insgesamt 392 Regeln aus unterschiedlichen Quellen:

- Microsoft Developer Network (MSDN) [2]
- Open Web Application Security Project (OWASP) [3]
- Computer Emergency Response Team (CERT) [4]
- verschiedene Publikationen

Sicherlich gibt es nicht die “eine” Quelle, die alle bekannten Schwachstellen dokumentiert. Da der Hersteller keine Angaben darüber macht, welche Abdeckung der SecurityChecker in Bezug auf die oben genannten Quellen erreicht, muss der Interessent und potentielle Käufer der Software selbst entscheiden, ob der Leistungsumfang des Werkzeugs ausreichend ist. Eine 14-Tage-Evaluationsversion kann unter [1] heruntergeladen werden.

Eine gute Referenz für dokumentierte Vulnerabilities und Exploits ist die US-CERT Vulnerability Notes Database ([5]). Die CERT besteht aus mehreren Organisationen, die sich mit Computersicherheit befassen, Sicherheitslücken dokumentieren und über Mailing-Listen informieren. Die US-CERT ermöglicht eine Online-Suche nach bekannten Sicherheitslücken über die Kriterien: Name, ID, CVE-Name, Härtegrad und

verschiedenen Datumsangaben. Zum aktuellen Zeitpunkt werden über 1.750 Schwachstellen unterschiedlichster Software-Produkte und -Technologien verwaltet.

Als Nachteil des SecurityCheckers muss in jedem Fall die Ausrichtung des Werkzeugs genannt werden: mit dem SecurityChecker unterstützt Compuware ausschließlich den Entwicklungsprozess. Die zu analysierende ASP.NET-Applikation muss inklusive Visual Studio auf dem Rechner installiert sein, auf dem die Sicherheitsanalyse durchgeführt werden soll. Somit kann das Tool nur auf einer Entwicklungsumgebung eingesetzt werden und ist weder für Tester noch für Administratoren der Produktionsumgebung nutzbar. (Der Autor dieses Artikels glaubt an das Gute und hofft, dass kein Leser des dot.net magazins je auf die Idee kommen wird, Entwicklungswerkzeuge auf einem Produktionsrechner zu installieren!) Der Anwender des Werkzeugs, d.h. der Entwickler der ASP.NET-Applikation, muss sich darüber im Klaren sein, dass er die Sicherheit der Applikation auf seinem Entwicklerrechner verifiziert. Ein Vulnerability Scanning auf einem Test- oder Produktionsserver ist mit dem SecurityChecker nicht möglich, daher sind für konkrete Entwicklungsprojekte bei einer Sicherheitsanalyse weitere Konkurrenzprodukte zu evaluieren. Die folgende Liste kann dabei als Orientierungshilfe dienen:

- Security Administrator's Integrated Network Tool (SAINT)
- ISS Security Scanner
- Nessus
- GFI LANGuard
- Security Administrator's Tool for Analyzing Networks (SATAN)
- Retina
- NIKTO
- SAFEsuite Internet Scanner
- Acunetix Web Vulnerability Scanner

Insgesamt ist den Werkzeugen im Bereich Vulnerability Scanning grundsätzlich vorzuwerfen, dass sich solche Tools ausschließlich auf Web-Applikationen fokussieren. Phishing-Mails und Angriffe auf Büronetzwerke bergen die Gefahr, dass Hacker sich einen Zugriff auf die intern eingesetzte Software des Unternehmens verschaffen und deren Arbeitsweise analysieren. Dabei ist eine Vielzahl der überprüften Regeln auch auf alle andere Projekttypen anwendbar (z.B. Windows Application, Class Library, Windows Control Library, Web Control Library, Console Application, Windows Service, Device Application, Web-Service).

Nüchtern fällt bei der Sicht ins Handbuch desweiteren auf, welche Neuerungen in Version 2.0 gegenüber der Vorversion 1.0 in den SecurityChecker eingebracht wurden:

- Support für Visual Studio 2005
- neue Security Rules für den Integrity Analyzer
- zusätzliche Informationen über Vulnerabilities
- Verbesserungen an den Discovery Maps
- eine neue Version des Distributed License Management Tools

Da sich die Stärke eines Vulnerability Scanners in dem Umfang der implementierten Regeln ausdrückt, wird der Ausbau des Regelwerks in der Compile-Time-Analyse und der Runtime-Analyse vermisst. Der Käufer eines Updates ist sicherlich nicht an Verbesserungen des Lizenz-Management-Tools interessiert.

Der Compuware DevPartner SecurityChecker kann unter [1] bezogen werden. Kasten 3 fasst nochmals die erforderlichen Hardware- und Softwarevoraussetzungen zusammen.

Kasten 3: Hardware- und Software-Voraussetzungen

Als Hardware ist ein herkömmlicher Entwicklungsrechner ausreichend:

- Pentium III >= 1.5 GHz
- RAM >= 1 GB
- Hard-Disk >= 400 MB

Als Software wird vorausgesetzt:

- Windows Server 2003 Standard, Web, Enterprise mit SP 1 und IIS 6.0
- Windows XP Professional Edition mit Service Pack 2 und IIS 5.1
- Windows 200 Professional, Server, Advanced Server/SP4 und IIS 5.0
- Visual Studio 2005 (VB.NET, C#.NET, C++.NET)
- Visual Studio .NET 2003 (VB.NET, C#.NET, C++.NET)

Manu Carus ist auf die Sicherheit von Anwendungen und Netzwerken spezialisiert. Sie erreichen ihn unter manu.carus@ethical-hacking.de.

Links & Literatur

[1] Compuware DevPartner SecurityChecker:
http://www.compuware.com/products/devpartner/securitychecker.htm
[2] Microsoft Developer Network (MSDN): http://msdn.microsoft.com/
[3] Open Web Application Security Project (OWASP): http://www.owasp.org/
[4] CERT Coordination Center (CERT): http://www.cert.org/
[5] US-CERT Vulnerability Notes Database: http://www.kb.cert.org/vuls/